WFAPI SDK Programmer's Guide

Introduction

The Citrix WinFrame API (WFAPI) functions enable application programs to perform tasks that are unique to Citrix Virtual Apps and Desktops.. Appropriate hotfixes and service packs may be needed for the functions defined here to execute properly.
Citrix Virtual Apps and Desktops server will be referred to as Citrix VDA in the Citrix WinFrame API documentation. Citrix Virtual Apps and Desktops (RDS VDA) runs on server OSs, Microsoft Windows 2019, Microsoft Windows 2022. Citrix Virtual Apps and Desktops (WS VDA) runs on workstation OSs, Windows 10, and Windows 11.
This SDK allows software developers to programmatically access features specific to Citrix VDAs. For example:

- Enumerating servers, sessions, and processes
- Managing servers, sessions, and processes
- Accessing Citrix-specific user data
- Sending messages to a session
- Using virtual channels
- Waiting on system events

The WFAPI SDK comprises a set of function calls to the Wfapi.dll dynamic link library (DLL) for 32-bit applications and Wfapi64.dll for 64-bit applications. These DLLs are available on servers and Windows Workstation platforms running Citrix Virtual Apps and Desktops. Example programs are included to assist developers.

Using the WFAPI SDK

The WFAPI SDK is intended for use by OEMs and customers who need to write applications that directly call WFAPI functions, using Microsoft Version 14.0 (Visual Studio 2015-2022).

System Requirements

The WFAPI SDK must be installed and built on a Windows 10, Windows 11, Windows Servers 2019 and 2022 computer with sufficient disk space. The SDK does not need more than a few megabytes of disk space. However, third-party development tools (for example, Microsoft Visual C++) may require a substantial amount of disk space. When considering system requirements for using this SDK, plan to have enough disk space for all of the necessary software components to install and function properly.

The WFAPI SDK is supported on 32-bit and 64-bit operating systems. The descriptions in this guide are primarily based on 64-bit operating systems and differences for 32-bit operating systems are noted.

Generally, wfapi64.lib and wfapi64.dll are needed if you intend to develop a genuine 64-bit application. If you do not intend to develop genuine 64-bit applications, you do not need to use the wfapi64.lib and wfapi64.dll files.

The WFAPI SDK has been upgraded to work with Microsoft Visual C++ Version 14.0. This SDK was built and tested with earlier versions of MS Visual Studio compilers (i.e. Visual Studio 8) for backward

compatibility. Install Visual C++ before installing the WFAPI SDK.

Execution Environment

Citrix Product	**Minimum Version Requirement**
Citrix Workspace App	Version 1912
Citrix Virtual Apps and Desktop	Version 1912

Required hotfixes, service packs, and clients are available from http://www.citrix.com/support.

Directory Layout

The following folders are created in the WFAPI SDK installation directory:

- DOCS – WFAPI SDK Guide and Readme
- EXAMPLES – C examples
- INCLUDE – C header file Wfapi.h
- LIB – C library files Wfapi.lib (for 32-bit applications) and Wfapi64.lib (for 64-bit applications)

Remarks:
Use lib inside the TS folder if you are trying to develop an application for server OS and use the lib inside the WS folder for workstation OS.

To install the WFAPI SDK

Run WFApiSDK.msi. Follow the installation dialogs. The default installation directory is %SystemDrive%\Program Files (x86)\Citrix\WFApiSDK.

To uninstall the WFAPI SDK

Use the Add or Remove Programs utility in the Windows Control Panel.

Example Programs

The WFAPI SDK includes C example programs that demonstrate how to use the WFAPI functions.

There are two C example programs: TESTALL and SMCCONSOLE. Both ANSI (Testall.exe) and Unicode (Testallw.exe and SmcConsole.exe) versions of the programs are supplied with each located in its own sub-directory. The makefile that is supplied with the example programs builds both the ANSI and Unicode versions of the executables. The Unicode version is made by defining the Unicode macros. You may modify these makefiles for your system as needed.

These example programs are written to run on 32-bit machines. To run these examples on a 64-bit machine, modify the makefiles so they link to wfapi64.lib instead of wfapi.lib.

The Testall and SMCConsole sample programs have been tested for use with Servers running RDS VDAs and Workstations running WS VDAs.

C Example Programs

The following instructions apply to Microsoft Visual Studio 2019. Adapt the instructions as needed for other environments.

To build the C example programs from a command prompt

1. If the INCLUDE and LIB environment variables are not set (type set at a command prompt to view them),open the developer command prompt by going to StartMenu search for Visual Studio 2019 and then open x86 Native Tools Command Prompt for VS2019 or x64 Native Tools Command Prompt for VS2019 as administrator depending on the type of build you want.
2. Set the environment variables LIB and INCLUDE using the below commands.
3. set LIB=%LIB%;C:\Program Files (x86)\Citrix\WfApiSDK\lib
4. set INCLUDE=%INCLUDE%;C:\Program Files (x86)\Citrix\WfApiSDK\include;C:\Program Files (x86)\Citrix\WfApiSDK\lib
5. Here C:\Program Files (x86)\Citrix\WfApiSDK is the path of the installed WfApi Sdk on your computer.
6. Run NMAKE on the desired directory.

Note: With the files as shipped, running NMAKE with no parameters will not rebuild the source files. To force rebuilding all files, run NMAKE CLEAN ALL.

To build the C example programs from within Microsoft Visual Studio 2019

1. Create a new project.
    a. Open a New Project. Select New from the File menu and select Project. Alternatively, press Ctrl+N or click New Project on the Start page.
    b. In the New Project window, select Visual C++ Projects in the left pane under Project Types.
    c. Select 'Empty Project' Project under Templates in the right pane.
    d. Type a project name in the Name text box (for example, wfapi).
    e. Enter the location of the project in the Location box or browse to a location.
    f. Select Application Settings to display the application settings window and configure the project.
    g. Select an option under Application type.
    h. Select Empty project under Additional options.
    i. Click Finish.
2. Add the source file (Testall.c) to the project.
    a. Select Add Existing Item from the Project menu or press Ctrl+Shift+D.
    b. In the Add Existing Item - wfapi window, enter a file name in the File name box. Alternatively, you can use the tool bar to find a file that you want to add.
    c. Select the file. Click Open.

3. Configure the build environment.
    a. Select Properties from the Project menu. The wfapi Property Pages window appears.
    b. Select All Configurations from the Configurations list. A tree of configuration settings appears in the left pane.
    c. Expand the C/C++ node.
    d. Select the General node.
    e. Enter the path for the Wfapi.h file in the Additional Include Directories box in the right pane.
    f. Expand the Linker node in the left pane.
    g. Select the General node.
    h. Select the Additional Library Directories box in the right pane. Enter the path for the Wfapi.lib file.
    i. Select 'Input' tab and then in the 'Additional Dependencies' Add the libraries 'wfapi.lib' and 'netapi32.lib'
    j. Click OK.
4. To build the Unicode version:
    a. Select Properties from the Project menu.
    b. Select All Configurations from the Configurations list.
    c. Expand the C/C++ node on the left.
    d. Select the General category.
    e. In the Preprocessor Definitions text box, type /DUNICODE and /D_UNICODE.
5. Build the executable. Select Build *Project* from the Build menu, where *Project* is the name of your project (for example, wfapi).

TESTALL

This example program uses a text menu interface to call most of the WFAPI functions. To retrieve information, specify the user name, server, and domain. These values affect the results of the WFQueryUserConfig() and WFSetUserConfig() functions.

Programming Guide

When you write Windows applications using the Microsoft Windows SDK, call WFAPI functions only when necessary. Your C applications can check for the presence of Wfapi.dll (for 32-bit applications) or Wfapi64.dll (for 64-bit applications), enabling them to run on any Windows 7 or greater Workstation or Server environment. These options help make your applications as portable as possible.

The server and workstation are used interchangeably in the below functions.

Function Summary

C Function	Description
WFCloseServer()	Closes an open server handle
WFDisconnectSession()	Disconnects a specified session
WFDisconnectSessionEx()	Forces an immediate disconnect of a specified session
WFEnumerateProcessesA() WFEnumerateProcessesW()	Retrieves a list of processes on the specified server
WFEnumerateProcessesExA() WFEnumerateProcessesEXW()	Retrieves a list of processes on a specified server
WFEnumerateSessionsA() WFEnumerateSessionsW()	Retrieves a list of sessions on a specified server
WFFreeMemory()	Frees memory allocated by WFAPI functions
WFLogoffSession()	Logs off a specified session
WFOpenServerA() WFOpenServerW()	Opens a handle to a specified server
WFQuerySessionInformationA()	Retrieves information about a specified session on a server

C Function	Description
WFQuerySessionInformationW()	
WFQueryUserConfigA() WFQueryUserConfigW()	Retrieves configuration information about a specified user on a server
WFSendMessageA() WFSendMessageW()	Sends a message box to a specified session
WFSetUserConfigA() WFSetUserConfigW()	Modifies configuration information for a specified user on a server
WFShutdownSystem()	Shuts down (and optionally restarts) the current server
WFTerminateProcess()	Terminates a specified process on a server
WFVirtualChannelClose()	Closes an open virtual channel handle
WFVirtualChannelOpen()	Opens a handle to a specific virtual channel
WFVirtualChannelPurgeInput()	Purges all queued input data sent from the client to the server on a specified virtual channel
WFVirtualChannelPurgeOutput()	Purges all queued output data sent from the server to the client on a specified virtual channel
WFVirtualChannelQuery()	Queries data related to a virtual channel
WFVirtualChannelRead()	Reads data from a virtual channel
WFVirtualChannelReadEx()	Reads data from a virtual channel
WFVirtualChannelWrite()	Writes data to a virtual channel
WFWaitSystemEvent()	Waits for an event (for example, ICA session create, delete, connect) before it returns
WFWaitSystemEvent()	Verify the ability to wait on system events like Connect/Disconnect/LogOFF/LogON/Shutdown;

Function Categories

The functions in the WFAPI SDK are grouped into several categories: internal, enumeration, server system, user, session, and virtual channel.

Internal Functions

The WFAPI SDK internal functions are:

- WFOpenServerA() and WFOpenServerW()
- WFCloseServer()
- WFFreeMemory()

These functions are internal to the WFAPI; they do not act upon the server or session. Most functions require a handle to a server. A handle is obtained using the WFOpenServerA()/WFOpenServerW() function. The constant WF_CURRENT_SERVER_HANDLE always refers to the current server and can be used without explicitly issuing WFOpenServerA()/WFOpenServerW().

Issue the WFCloseServer() function as part of your program's clean-up when you are finished with the handle. Do not close WF_CURRENT_SERVER_HANDLE.

If you call a C function that allocates and returns memory, de-allocate the memory using the WFFreeMemory() function when you are finished.

Enumeration Functions

The WFAPI SDK enumeration functions are:

- WFEnumerateProcessesA() and WFEnumerateProcessesW()
- WFEnumerateProcessesExA() and WFEnumerateProcessesExW()
- WFEnumerateSessionsA() and WFEnumerateSessionsW()

These functions are used to enumerate processes and sessions. The WFEnumerate Processes, WFEnumerateProcessesEx, and WFEnumerateSessions functions require a server handle.

The WFEnumerateProcessEx functions return more detailed information than the WFEnumerateProcesses functions.

Server System Functions

The WFAPI SDK server system functions are:

- WFShutdownSystem()
- WFWaitSystemEvent()

These functions are used to manage a server system. The WFShutdownSystem() and WFWaitSystemEvent() functions require a server handle. WFShutdownSystem() shuts down and optionally restarts the current server, and can be used to disable logons. WFWaitSystemEvent() waits for an event (ICA session create/delete/connect, user logon/logoff, and so on) before it returns.

User Functions

The WFAPI SDK user functions are:

- WFQueryUserConfigA() and WFQueryUserConfigW()
- WFSetUserConfigA() and WFSetUserConfigW()

These functions are used to retrieve or set user configuration information. Both functions require a server name, a user name, and the class of information that will be operated on. A handle is not required. The constant WF_CURRENT_SERVER_NAME refers to the current server.

These functions are passed a server name instead of a handle because user account information often resides on a domain controller.

- Use the Windows function NetGetDCName() to retrieve the name of the primary domain controller for the WFSetUserConfig() function.
- Use the Windows function NetGetAnyDCName() to retrieve the name of any domain controller for the WFQueryUserConfig() function.

You can use any domain controller to query domain information; however, you must use the primary domain controller when changing domain information. This is in agreement with other Microsoft API functions. The Visual Basic USER example program demonstrates the use of the NetGetAnyDCName() function.

Session Functions

The WFAPI SDK session functions are:

- WFQuerySessionInformationA() and WFQuerySessionInformationW()
- WFSendMessageA() and WFSendMessageW()
- WFDisconnectSession() and WFDisconnectSessionEx()
- WFLogoffSession()
- WFTerminateProcess()

These functions are used to operate on active sessions. All functions require a server handle and a session ID.

Virtual Channel Functions

The WFAPI SDK virtual channel functions are:

- WFVirtualChannelOpen()
- WFVirtualChannelClose()
- WFVirtualChannelRead()
- WFVirtualChannelReadEx()
- WFVirtualChannelWrite()
- WFVirtualChannelPurgeInput()
- WFVirtualChannelPurgeOutput()
- WFVirtualChannelQuery()

The ICA protocol provides multiplexed management of multiple virtual channels. A virtual channel is a

session-oriented transmission connection that can be used by application layer code. Virtual channels are used to add functional enhancements to the client, independent from the ICA protocol.

The WFVirtualChannelOpen() function requires a server handle, a session ID, and a virtual channel name. It returns a virtual channel handle. The other functions require the resulting virtual channel handle.

Use the WFVirtualChannelClose() function to close the handle when it is no longer needed.

Unicode and ANSI Versions

Most functions have both Unicode and ANSI versions. ANSI is an older, more common standard, but is difficult to use simultaneously with multiple languages. Unicode is a 16-bit (wide) character set that supports up to 65,536 unique characters, allowing for the representation of multiple language characters using a single character set.

For each function that has both ANSI and Unicode declarations, both declarations are listed under the main heading. Each function is declared as either *functionname*A( ) for the ANSI version or *functionname*W( ) for the Unicode (wide) version.

In C, the additional #define for *functionname*() allows you use a compiler directive to use all ANSI or all Unicode functions in the resulting executable. This makes it easy to produce both ANSI and Unicode versions of your programs because you need to change only the makefile.

In Visual Basic, explicit function names must be used for those functions that have both ANSI and Unicode versions defined.

C API Calling Conventions

C-language programs that use the WFAPI functions must include Wfapi.h, and link to Wfapi.lib for 32-bit applications or Wfapi64.lib for 64-bit applications.

The words IN and OUT that are listed in the function calling conventions are included only for clarification. They are defined as blank in the Windef.h file. If you do not have access to Windef.h, add the following lines to the header file for your project:

```
#ifndef IN
#define IN
#endif
#ifndef OUT
#define OUT
#endif
```

WINAPI, as listed in the function calling conventions, is defined in the Windows.h file. If your program uses dynamic linking with runtime binding, you must include Windows.h in the function pointer definition. For more information, see the Testall.c example.

If a function succeeds, the return value is TRUE. If a function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Application Programming Guidelines

- Avoid using hard coded paths, such as C:\Temp.
- Use the Win32 API functions whenever possible. Many Windows API functions have Citrix enhancements to seamlessly support a multiuser environment.
- Do not assume that temporary files are persistent.
- Do not assume the existence of a default printer.
- Be sensitive to the bandwidth used by printing tasks.
- Do not use machine-specific attributes (such as machine names or IP addresses) for unique identifiers. These items are common to all users on each machine.
- Write user-specific settings to HKEY_CURRENT_USER, not HKEY_LOCAL_MACHINE.
- Do not allow “File,” “Run” capabilities.
- Limit necessary animation to reduce bandwidth consumption.
- Use solid-color graphics for better ICA compression.
- Do not overwrite Windows Terminal Services-specific DLLs with Windows DLLs.
- Do not try to allocate all available memory.
- Do not use continuous polling loops in your code because they require excessive CPU cycles. Use event-driven techniques instead.
- Thoroughly test required peripheral devices over all connection types.
- Do not assume that your application will be run in a Windows Explorer shell.
- Memory leaks and their negative effects are greatly compounded in a multiuser environment.
- Avoid using bitmaps in graphics; use vector-based graphics instead. For best performance on an ICA device, use the raster operator to “brush” graphics onto the screen.
- Some applications that use the TCP/IP protocol to communicate use the IP address as the hard-coded identifier of the client. Multiple instances of these applications will not run in a Citrix environment.
- For an application to properly communicate in a Citrix environment, the application must be able to negotiate a private socket, allowing the client and server to communicate using a unique IP/PORT/SOCKET address.
- The use of True Type fonts is preferred; these font types are stored on the client. If an application must use custom or Adobe fonts, configure them to be embedded Windows fonts for faster display.

Determining the Citrix VDA Version

To determine the version of a Citrix VDA, issue the WFQuerySessionInformation() function with WFInfoClass set to WFVersion. Here is a C code segment extracted from the Testall.c example program.

```
HANDLE hServer = WF_CURRENT_SERVER_HANDLE;
DWORD SessionId = WF_CURRENT_SESSION;
LPTSTR pSessionInfo;
DWORD ByteCount;
LPOSVERSIONINFO pVerInfo;

if ( !WFQuerySessionInformation( hServer,
                        SessionId,
                        WFVersion,
                        &pSessionInfo,
                         &ByteCount ) ) {
   _tprintf( TEXT("”***WFQuerySessionInformation failed, error %u\n"),
          GetLastError() );
   return;
}
pVerInfo = (LPOSVERSIONINFO) pSessionInfo;
_tprintf( TEXT("Version: major %u, minor %u, build %u, CSD: %s\n"),
            pVerInfo->dwMajorVersion, pVerInfo->dwMinorVersion,

            pVerInfo->dwBuildNumber, pVerInfo->szCSDVersion);
```

Results

If this function is successful, it will return specific information about the version of the Citrix VDA that is installed on your server. For example, if XenApp Server 6.0 is installed, this information will be displayed:

```
Major 6, minor 0, build 2198
```

Note: The results may vary, depending on the system configuration, and the hotfixes and service packs that are installed on the system.

Programming Reference

Unless otherwise specified the following functions apply to Citrix VDAs. In the context of this reference a 'server' can be any supported Windows OS that is running a Citrix VDA.

WFCloseServer

Close a server handle that was opened by WFOpenServer.

C Calling Convention

```
VOID WINAPI WFCloseServer(IN HANDLE hServerHandle );
```

hServerHandle

A handle to a previously-opened server. Use the WFOpenServer() function to obtain a handle for a specific server.

WFDisconnectSession

This function disconnects a specified session.

C Calling Convention

```
BOOL WINAPI WFDisconnectSession(
  IN HANDLE hServer,
  IN DWORD SessionId,
  IN BOOL bWait);
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a specific server or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of the target session. WF_CURRENT_SESSION identifies the current session. Use the WFEnumerateSessions() function to obtain session IDs.

bWait

If this value is TRUE, the function waits for the disconnect function to complete before it returns.

Return Values

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFDisconnectSessionEx

This function forces an immediate disconnect of a specified session.

C Calling Convention

```
BOOL WINAPI WFDisconnectSessionEx(
  IN HANDLE hServer,
  IN DWORD SessionId,
  IN ULONG Flags,
  IN PVOID pBuffer,
  IN ULONG BufferLength
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a specific server or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of the target session. WF_CURRENT_SESSION identifies the current session. Use the WFEnumerateSessions() function to obtain session IDs.

Flags

A combination of the following flags:

- WF_DISCON_WAIT. If this flag is set, the function waits for the disconnect to finish; if this flag is not set, the function does not wait for the disconnect to finish.
- WF_DISCON_NO_NOTIFY_CLIENT. If this flag is set, the function returns without notifying the client; if this flag is not set, the function waits for the client's response to ensure that the client received the disconnect request.

pBuffer

This parameter is reserved for future use.

BufferLength

This parameter is reserved for future use.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function forces an immediate disconnect of the session. Use the WF_DISCON_NO_NOTIFY_CLIENT flag with caution because it does not notify the client of the disconnect.

WFEnumerateProcesses

This function retrieves a list of the active processes on a specified server.

C Calling Convention

```
BOOL WINAPI WFEnumerateProcesses{W|A}(
  IN HANDLE hServer,
  IN DWORD Reserved,
  IN DWORD Version,
  OUT PWF_PROCESS_INFO{W|A} * ppProcessInfo,
  OUT DWORD * pCount
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a specific server or specify WF_CURRENT_SERVER_HANDLE to use the current server.

Reserved

This value must be zero (0).

Version

The version of the enumeration request, this value must be 1.

ppProcessInfo

A pointer to the address of a variable that will receive the enumeration results that are returned as an array of WF_PROCESS_INFO structures (defined in Wfapi.h). The buffer is allocated by this function and is de-allocated using the WFFreeMemory() function.

pCount

A pointer to the variable that will receive the number of returned WF_PROCESS_INFO structures.

The name of the process pointed to by pProcessName contains an ANSI string if WFEnumerateProcessesA() is used, and a Unicode string if WFEnumerateProcessesW() is used.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function only applies to RDS VDAs.

WFEnumerateProcessesEx

This function retrieves a list of the active processes on a specified server. Detailed process information is also returned.

C Calling Convention

```
BOOL WINAPI WFEnumerateProcessesEx{W|A}(
  IN HANDLE hServer,
  IN DWORD Reserved,
  IN DWORD Version,
  OUT PWF_PROCESS_INFOEX{W|A} * ppProcessInfo,
  OUT DWORD * pCount
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a specific server or specify WF_CURRENT_SERVER_HANDLE to use the current server.

Reserved

This value must be zero (0).

Version

The version of the enumeration request, this value must be 1.

ppProcessInfo

A pointer to the address of a variable that will receive the enumeration results that are returned as an array of WF_PROCESS_INFOEX structures (defined in Wfapi.h). The buffer is allocated by this function and is de-allocated using the WFFreeMemory() function.

pCount

A pointer to the variable that will receive the number of returned WF_PROCESS_INFOEX structures.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function only applies to RDS VDAs.

WFEnumerateSessions

This function retrieves a list of the sessions on a specified VDA.

C Calling Convention

```
BOOL WINAPI WFEnumerateSessions{W|A}(
  IN HANDLE hServer,
  IN DWORD Reserved,
  IN DWORD Version,
  OUT PWF_SESSION_INFO{W|A} * ppSessionInfo,
  OUT DWORD * pCount
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

Reserved

This value must be zero (0).

Version

The version of the enumeration request, this value must be 1.

ppSessionInfo

A pointer to the address of a variable that will receive the enumeration results that are returned as an array of WF_SESSION_INFO structures (defined in Wfapi.h). The buffer is allocated by this function and is de-allocated using the WFFreeMemory() function.

pCount

A pointer to the variable that will receive the number of returned WF_SESSION_INFO structures.

The session name pointed to by pWinStationName contains an ANSI string if WFEnumerateSessionsA() is used, and a Unicode string if WFEnumerateSessionssW() is used.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function applies to RDS VDAs; on WS VDAs it will only show the console session.

WFFreeMemory

This function releases memory allocated by WFAPI functions.

C Calling Convention

```
VOID WINAPI WFFreeMemory(
  IN PVOID pMemory
  );
```

pMemory

A pointer to the memory to be freed.

Remarks

Many functions allocate a data area to return information. Use this function to release these data areas.

WFLogoffSession

This function logs off a specified session.

C Calling Convention

```
BOOL WINAPI WFLogoffSession(
  IN HANDLE hServer,
  IN DWORD SessionId,
  IN BOOL bWait
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of the target session. WF_CURRENT_SESSION specifies the current session.

bWait

Specify TRUE to instruct the server to wait for the operation to complete. Specify FALSE to log off the session in the background. To verify that the session is logged off, use the WFQuerySessionInformation() function to query the session ID and check for a FALSE return value.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFOpenServer

This function opens a handle to a specified server.

C Calling Convention

```
HANDLE WINAPI WFOpenServer{W|A}(
  IN LPWSTR pServerName
  );
```

pServerName

A pointer to the name of a server that is running a VDA.

Supply an ANSI string to WFOpenServerA() and a Unicode string to WFOpenServerW().

Returns

If the function succeeds, it returns a handle to the specified server. If the function fails, it returns NULL. Call the GetLastError() function to get extended error status.

Remarks

When you are finished with the handle, use the WFCloseServer() function to close all server handles opened using the WFOpenServer function, as part of you program's cleanup.

You do not need to open a handle for operations performed on the server where the program is running. Use the constant WF_CURRENT_SERVER_HANDLE instead.

WFQuerySessionInformation

This function retrieves information about a specified session on a specified server.

C Calling Convention

```
BOOL WINAPI WFQuerySessionInformation{W|A}(
  IN HANDLE hServer,
  IN DWORD SessionId
  IN WF_INCO_CLASS WFInfoClass,
  OUT LPWSTR * ppBuffer,
  OUT DWORD * pBytesReturned
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of the target session. WF_CURRENT_SESSION specifies the current session.

WFInfoClass

Type of information to be retrieved from the specified session.

WFInfoClass value	**Data type**	**Note**
WFVersion	OSVERSIONINFO	1
WFInitialProgram	NULL-terminated string	
WFWorkingDirectory	NULL-terminated string	
WFOEMId	NULL-terminated string	
WFSessionId	ULONG	
WFUserName	NULL-terminated string	
WFWinStationName	NULL-terminated string	2
WFDomainName	NULL-terminated string	
WFConnectState	INT	2
WFClientBuildNumber	USHORT	3
WFClientName	NULL-terminated string	3
WFClientDirectory	NULL-terminated string	3
WFClientProductId	USHORT	3
WFClientAddress	WF_CLIENT_ADDRESS	2, 3
WFClientDisplay	WF_CLIENT_DISPLAY	2, 3
WFClientCache	WF_CLIENT_CACHE	2, 3

WFInfoClass value	Data type	Note
WFClientDrives	WF_CLIENT_DRIVES	2, 3
WFICABufferLength	ULONG	3
WFApplicationName	NULL-terminated string	3
WFAppInfo	WF_APP_INFO	2, 3
WFClientInfo	WF_CLIENT_INFO	2, 3
WFUserInfo	WF_USER_INFO	2, 3
WFSessionTime	WF_SESSION_TIME	2

- 1 - This is a standard Windows structure defined in Winbase.h. For more information, see the Windows programming documentation. This option returns information only for the current server; the server handle parameter is ignored for this option.
- 2 - These structures are defined in Wfapi.h.
- 3 - These parameters can be used only if the function is called from an ICA session. If run from the server console or a Direct ICA station, the function returns FALSE.
- 4 - The strings are in ANSI format if WFEnumerateSessionInformationA() is used, and Unicode format if WFEnumerateSessionInformationW() is used.
- 5 - Use an INTEGER or LONG to retrieve a SHORT.

ppBuffer

A pointer to the address of a variable that will receive the information about the specified session. The format and contents of the data depend on the specified information class being queried; see the second column of the table above. The buffer is allocated by this function and is de-allocated using the WFFreeMemory() function.

pBytesReturned

A pointer to the variable that will receive the number of returned bytes. If this value is NULL, the byte count is not returned.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

The WFSessionId WFInfoClass type returns zero (0) when running on the server console. This is the best method for determining if a program is being run from an ICA session or on the server console itself.

The WFVersion WFInfoClass type can be used only to retrieve operating system version information for the current server that is running XenApp Server. The server handle is ignored for this option.

On Workstation VDAs there is only one session, use WF_CURRENT_SESSION for the session Id.

WFQueryUserConfig

This function retrieves configuration information about a specified user on a server.

C Calling Convention

```
BOOL WINAPI WFQueryUserConfig{W|A}(
  IN LPWSTR pServerName,
  IN LPWSTR pUserName,
  IN WF_CONFIG_CLASS WFConfigClass,
  OUT LPWSTR * ppBuffer,
  OUT DWORD * pBytesReturned
  );
```

pServerName

A pointer to the name of a server or a domain controller. Use WF_CURRENT_SERVER_NAME to specify the current server.

The WFQueryUserConfig() and WFSetUserConfig() functions are passed a server name instead of a handle because user account information often resides on a domain controller. The domain controller can be a Windows server or a server. Any domain controller can be used to retrieve domain information, but you must use the primary domain controller to change domain information. Use the following standard Windows functions, which are in agreement with other Microsoft APIs.

- Use NetGetDCName() to get the name of the primary domain controller for the WFSetUserConfig() function
- Use NetGetAnyDCName() to get the name of any domain controller for the WFQueryUserConfig() function

pUserName

A pointer to the user name.

WFConfigClass

The type of user information to be retrieved.

WFConfigClass Value	**(Data type) and data returned**
WFUserConfigInitialProgram	(NULL-terminated string) User initial program
WFUserConfigWorkingDirectory	(NULL-terminated string) User working directory
WFUserConfigfInheritInitialProgram	(DWORD) Flag for inheriting initial program

ppBuffer

A pointer to the address of a variable that will receive the query results. The buffer is allocated within this function and is de-allocated using the WFFreeMemory() function.

pBytesReturned

A pointer to the variable that will receive the number of returned bytes.

If WFQueryUserConfigA() is used, the supplied server name must be an ANSI string. If WFQueryUserConfigW() is used, teh supplied server name must be a Unicode string.

The NetGetDCName() and NetGetAnyDCName() functions accept only Unicode strings as parameters.

The strings returned for the user initial program and user working directory are in ANSI format if WFQueryUserconfigA() is used, and Unicode format if WFQueryUserConfigW() is used.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function applies to RDS VDAs.

WFSendMessage

This function sends a message box to a specified session.

C Calling Convention

```
BOOL WINAPI WFSendMessage{W|A}(
  IN HANDLE hServer,
  IN DWORD SessionId,
  IN LPWSTR pTitle,
  IN DWORD TitleLength,
  IN LPWSTR pMessage,
  IN DWORD MessageLength,
  IN DWORD Style,
  IN DWORD Timeout,
  OUT DWORD * pResponse,
  IN BOOL bWait
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of a session that is running a Citrix VDA. Use WF_CURRENT_SESSION to specify the current session.

pTittle

A pointer to the title that the message box will display.

TitleLength

The length (in bytes) of the title to be displayed.

pMessage

A pointer to the message to be displayed.

MessageLength

The length (in bytes) of the message to be displayed.

Style

A standard Windows MessageBox() style parameter. For more information, see the Microsoft documentation. Typically, this value is set to MB_OK for C++.

Timeout

The response time-out, in seconds. If the message does not receive a response within this time period and *bWait* is set to TRUE, *pResponse* contains IDTIMEOUT.

bWait

If TRUE, the function waits for the user to respond or the time-out to occur before returning control. If no *Timeout* value is specified, the user must respond before control is returned.
If FALSE, control is returned immediately and *pResponse* contains IDASYNC. Use this method for simple information messages (such as print job notification messages) that do not need to return the user's response to the calling program.

pResponse

A pointer to a variable that indicates the location where the selected response will be returned. The response is one of: a MessageBox() return code, IDASYNC, or IDTIMEOUT.

If WFSendMessageA() is used, the supplied title and message must be ANSI strings. If WFSendMessageW() is used, the title and message must be Unicode strings.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFSetUserConfig

This function modifies configuration information for a specified user on a server or domain controller.

C Calling Convention

```
BOOL WINAPI WFSetUserConfig{W|A}(
  IN LPWSTR pServerName,
  IN LPWSTR pUserName,
  IN WF_CONFIG_CLASS WFConfigClass,
  IN LPWSTR * pBuffer,
  IN DWORD DataLength
  );
```

pServerName

A pointer to the name of a server or a domain controller. Use WF_CURRENT_SERVER_NAME to specify the current server.

The WFQueryUserConfig() and WFSetUserConfig() functions are passed a server name instead of a handle because user account information often resides on a domain controller. The domain controller can be a Windows server or a server that is running a Citrix VDA. Any domain controller can be used to retrieve domain information, but you must use the primary domain controller to change domain information. Use the following standard Windows functions, which are in agreement with other Microsoft APIs.

- Use NetGetDCName() to get the name of the primary domain controller for the WFSetUserConfig() function
- Use NetGetAnyDCName() to get the name of any domain controller for the WFQueryUserConfig() function

pUserName

A pointer to the user name.

WFConfigClass

The type of user information to be set.

WFConfigClass Value	**(Data type) and data returned**
WFUserConfigInitialProgram	(NULL-terminated string) User initial program
WFUserConfigWorkingDirectory	(NULL-terminated string) User working directory
WFUserConfigfInheritInitialProgram	(DWORD) Flag for inheriting initial program

pBuffer

A pointer to the data that is used to modify the configuration of the specified user.

DataLength

The size (in bytes) of the buffer.

The strings returned for the user initial program and user working directory are in ANSI format if WFQueryUserconfigA() is used, and Unicode format if WFQueryUserConfigW() is used.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

You must have domain administrator privileges to use this function. This function applies to RDS VDAs.

WFShutdownSystem

This function shuts down (and optionally restarts) the current server.

C Calling Convention

```
BOOL WINAPI WFShutdownSystem(
  IN HANDLE hServer,
  IN DWORD ShutdownFlag
  );
```

hServer

This parameter is ignored.

ShutdownFlag

One of the following flags. Flags cannot be combined.

Shutdown Flag	Description
WF_WSD_LOGOFF	Forces all sessions except the console to log off and disables all further logons. Use this flag only on the server console. If the flag is used from another session, all sessions are logged off and no further logons are allowed. If a user is logged onto the console when the flag is used, the console is not logged off.
WF_WSD_SHUTDOWN	Shuts down the pc. This action is equivalent to selecting Shutdown from the Start menu or Program Manager. A dialog box appears on the server console, prompting the user to press a key to restart the pc.
WF_WSD_REBOOT	Shuts down and restarts the pc. This action is equivalent to selecting Shutdown and Reboot from the Start menu or Program Manager.
WF_WSD_FASTREBOOT	Restarts the server without shutting it down. This action is equivalent to running the SHUTDOWN / NOW command, which is typically not done. For more information, see the Remarks section.
WF_WSD_POWEROFF	Shuts down the pc and, on computers that support software control of power, powers it off.

Returns

When this function is run on the server console, if the WF_WSD_LOGOFF flag is used and the function succeeds, the return value is TRUE. If this function is run on another session, or one of the other flags is used and the function succeeds, the program terminates and the pc is shut down.

If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended

error status.

Remarks

A system shutdown terminates all users and active programs. Users are not warned before they are logged off and they do not have the option of saving their work. The following steps occur during shutdown:

1. An exit command is issued to all active user applications. If an application does not exist within a specified interval, the application is terminated.
2. After all of the applications for a user terminate, the user is logged off.
3. After all users are logged off, an exit command is issued to all system services. If a system service does not terminate within a specified interval, the service is terminated.
4. After all system services are terminated, the file system cache is written to disk.
5. The disks are marked “Read Only.”
6. If the WF_WSD_REBOOT flag is specified, the system is restarted. If not, the System Can Be Turned Off message appears.

Caution: WF_WSD_FASTREBOOT forces an immediate shutdown, skipping Steps 1 through 3 above. Do not use WF_WSD_FASTREBOOT unless it is absolutely necessary because there is a possibility of severe data loss or file corruption.

Because there can be many users and processes in a large multiuser configuration, large system configurations may take some time to shut down properly. It is important to allow the system to shut down completely.

WFTerminateProcess

This function terminates a specified process on a server.

C Calling Convention

```
BOOL WINAPI WFTerminateProcess(
  IN HANDLE hServer,
  IN DWORD ProcessId,
  IN DWORD ExitCode
  );
```

hServer

A handle to a pc that is running a VDA. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

ProcessId

The process ID of the process to be terminated.

ExitCode

The termination status for the process.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function applies to RDS VDAs.

WFVirtualChannelClose

This function closes an open virtual channel handle.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelClose(
  IN HANDLE hChannelHandle
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelOpen

This function opens a handle to a specific virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelOpen(
  IN HANDLE hServer,
  IN DWORD SessionId,
  IN LPSTR pVirtual Name    /* ANSI name */
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

SessionId

The session ID of the target session. Use WF_CURRENT_SESSION to specify the current session.

pVirtualName

A pointer to the virtual channel name. This is a seven-character ANSI string (even when Unicode is defined), padded with trailing spaces if necessary.

The first three characters represent the OEM ID (CTX for Citrix Systems, Inc.). The next four characters represent the virtual channel name.

The currently-defined virtual channels are:

Defined name	**ANSI string**	**Description**
VIRTUAL_CDM	"CTXCDM "	Client drive mapping
VIRTUAL_THINWIRE	"CTXTW "	Remote window data

Note: The virtual channel name must be a string exactly 8 bytes long. The number of visible characters is 7. The last byte is 0, which is used to terminate the string. Note the padding spaces in each name.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

This function can be used only with existing virtual channels. To develop virtual channel drivers, use this SDK and the Citrix Virtual Channel SDK. On WS VDAs there is only one session, use WF_CURRENT_SESSION to specify the target session.

** To allow all virtual channels to be opened inside an ICA session we need to disable the 'Virtual Channel Allow list' policy on DDC.

WFVirtualChannelPurgeInput

This function purges all queued input data sent from the client to the server on a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelPurgeInput(
  IN HANDLE hChannelHandle
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelPurgeOutput

This function purges all queued output data sent from the server to the client on a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelPurgeOutput(
  IN HANDLE hChannelHandle
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelQuery

This function retrieves data that is related to a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelQuery(
  IN HANDLE hChannelHandle,
  IN WF_VIRTUAL_CLASS VirtualClass,
  OUT PVOID *ppBuffer,
  OUT DWORD *pBytesReturned
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

VirtualClass

The type of information that will be requested. Currently, the only defined value is WFVirtualClientData, which returns virtual channel client module data.

ppBuffer

A pointer to the address of a variable that will receive the data. The buffer is allocated within this function and is de-allocated using the WFFreeMemory() function.

pBytesReturned

A pointer to the variable that is updated with the length of the data that is returned in the allocated buffer upon a successful return.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelRead

This function reads data from a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelRead(
  IN HANDLE hChannelHandle,
  IN ULONG TimeOut,
  OUT PCHAR Buffer,
  IN ULONG BufferSize,
  OUT PULONG pBytesRead
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

TimeOut

The time-out value in milliseconds. If this value is zero (0), the function returns immediately because there is no data to be read. If the value is INFINITE (or -1), the function continues to wait until there is data to be read. (INFINITE is defined in the Winbase.h file; for more information, see the Microsoft programming documentation.)

Buffer

The buffer that will receive the data that is read from the virtual channel.

BufferSize

The length, in bytes, of the receive buffer.

pBytes

A pointer to a variable that will receive the number of bytes read.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelReadEx

This function reads data from a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelReadEx(
  IN HANDLE hChannelHandle,
  IN HANDLE hCancelEvent,
  OUT PCHAR Buffer,
  IN ULONG BufferSize,
  OUT PULONG pBytesRead
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

hCancelEvent

A handle to an event upon which the function will wait to cancel the read when it is signaled. The event must exist before this function is issued.

Buffer

The buffer that will receive the data that is read from the virtual channel.

BufferSize

The length, in bytes, of the receive buffer.

pBytes

A pointer to a variable that will receive the number of bytes read.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFVirtualChannelWrite

This function writes data to a specified virtual channel.

C Calling Convention

```
BOOL WINAPI WFVirutalChannelWrite(
  IN HANDLE hChannelHandle,
  IN PCHAR Buffer,
  IN ULONG Length,
  OUT PULONG pBytesWritten
  );
```

hChannelHandle

A handle to a previously-opened virtual channel. Use the WFVirtualChannelOpen() function to obtain a handle for a specific channel.

Buffer

The buffer that contains the data to be written to the virtual channel.

Length

The length, in bytes, of the data to be written.

pBytesWritten

A pointer to a variable that will receive the number of bytes written.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

WFWaitSystemEvent

This function waits for an event (ICA session create/delete/connect, user logon/logoff, and so on) before it returns.

C Calling Convention

```
BOOL WINAPI WFWaitSystemEvent(
  IN HANDLE hServer,
  IN DWORD EventMask,
  OUT DWORD * pEventFlags
  );
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server.

EventMask

The bit mask that specifies the event(s) for which the function will wait. These values can be combined to wait for the first of multiple events.

pEventFlags

A pointer to a variable that will receive a bit mask of events that occurred.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. Use the GetLastError() function to get the extended error status.

Remarks

Citrix VDA creates ICA connections dynamically as needed. The following table lists and describes the events (possible values for *EventMask*), and indicates the flags triggered by the event.

Event	**Description**	**Flags triggered**
WF_EVENT_CREATE	New ICA session created	Create, State Change, All
WF_EVENT_DELETE	Existing ICA session deleted	Delete, State Change, All
WF_EVENT_LOGON	User logon to system (from console or WinStation)	Logon, State Change, All
WF_EVENT_LOGOFF	User logoff from system (from console or WinStation)	Delete, Logoff, State Change, All
WF_EVENT_CONNECT	ICA session connect from client	Connect, State Change, All
WF_EVENT_DISCONNECT	ICA session disconnect from client	Disconnect, State Change, All

Event	Description	Flags triggered
WF_EVENT_RENAME	Existing ICA session renamed	Rename, All
WF_EVENT_STATECHANGE	ICA session state change (this event is triggered when WF_CONNECTSTATE_CLASS (defined in Wfapi.h) changes)	
WF_EVENT_LICENSE	License state change (this event is triggered when a license is added or deleted using License Manager)	License, All
WF_EVENT_ALL	Wait for any event type	
WF_EVENT_FLUSH	Unblock all waiting events (this event is used only as an *EventMask*)	
WF_EVENT_NONE	No event (this event is used only as a return value in *pEventFlags*)	

An ICA connection is created when a user connects. When a user logs off, the ICA session is deleted.

When a user logs on to a disconnected session, the existing session is deleted and the Delete flag is triggered. When users connect to a disconnected session from within a session, their session is disconnected and the Disconnect flag is triggered instead of the Delete flag.

**This API produces inconsistent results for WS VDA's.

WFGetCtxSessionKeyList

Retrieve a list of Session Keys

C Calling Convention

```
DWORD WINAPI WFGetCtxSessionKeyList(HANDLE hServer, WCHAR **SessionKeys, DWORD
                                    *KeyCount, DWORD *dwKeyBufferLen, DWORD *KeyStrLen)
```

hServer

A handle to a server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server

SessionKeys

A pointer to a buffer that will receive an array of session keys, which is allocated upon successful return.

KeyCount

A pointer to a DWORD that will be set to the count of SessionKeys.

dwKeyBufferLen

Pointer to a DWORD which is updated with the length of the data returned in the allocated buffer upon successful return.

KeyStrLen

Length of the stings returned. The strings are GUIDS so their length will be equal across all session keys.

Returns

Return is ERROR_SUCCESS if successful otherwise an error code. If successful, the caller is responsible for deallocating the buffer returned.

Remarks

Shows 0 session keys for Workstation VDA's.

WFIsThisDoubleHopSession

This function returns TRUE if the session is an ICA double hop else returns FALSE.

hServer

A handle to a server that is running XenApp Server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server. Currently WF_CURRENT_SERVER_HANDLE is supported

SessionId

A XenApp Server session ID. WF_CURRENT_SESSION identifies the current session. Currently current sessionId or WF_CURRENT_SESSION supported.

Remarks

A double hop session is an ICA session running from an ICA session. This API must be called from the client.

WFWaitEndPointClientEventInDoubleHop

This function waits for an endpoint event (ICA session connect/disconnect) on a first-hop ICA session before it returns.

C Calling Convention

BOOL WINAPI **WFWaitEndPointClientEventInDoubleHop**(

IN HANDLE hServer,

IN DWORD EventMask,

OUT DWORD * pEventFlags

);

hServer

A handle to a server that is running XenApp Server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server. Currently WF_CURRENT_SERVER_HANDLE is supported

EventMask

The bit mask that specifies the event(s) for which the function will wait. These values can be combined to wait for the first of multiple events.

pEventFlags

A pointer to a variable that will receive a bit mask of events that occurred.

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Remarks

XenApp Server creates ICA connections dynamically as needed. The following table lists and describes the events (possible values for EventMask), and indicates the flags triggered by the event.

Event	Description	Flags triggered
WF_EVENT_CONNECT	ICA session connect from Endpoint to first-hop	Connect
WF_EVENT_DISCONNECT	ICA session connect from Endpoint to first-hop	Disconnect

** API fails to wait if the second hop is WS-VDA

WFEndPointClientDataInDoubleHopW

Provides the Client name, ipaddress, productID and build number described in **WF_CLIENT_INFOW** structure if the session is running in a double-hop session.

BOOL WINAPI **WFEndPointClientDataInDoubleHopW**(

IN HANDLE hServer

IN DWORD SessionId,

IN PWF_CLIENT_INFOW* ppClientInfo,

DWORD* pDataSize);

hServer

A handle to a server that is running XenApp Server. Use the WFOpenServer() function to obtain a handle for a particular server, or specify WF_CURRENT_SERVER_HANDLE to use the current server. Currently WF_CURRENT_SERVER_HANDLE is supported

SessionId

A XenApp Server session ID. WF_CURRENT_SESSION identifies the current session. Currently current sessionId or WF_CURRENT_SESSION supported.

ppClientInfo

A pointer to the address of a variable type **WF_CLIENT_INFOW** that will receive the information about the endpoint client data. The buffer is allocated by this function and is deallocated using the WFFreeMemory() function.

pDataSize

A pointer to the variable that will receive the number of returned bytes. If this value is NULL, the byte count is not returned

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

WFCaptureAppDisplay

Captures the contents of an application window which can be consumed by virtual channels using the VC SDK.

BOOL WINAPI **WFCaptureAppDisplay**(

IN HWND hwnd

IN AppDisplayCallback cb

IN PVOID userdata

IN UINT32 flags);

hWnd
The window handle of the application whose display is to be captured.

cb
A callback to receive events.

userdata
A pointer to an application context or other user data that will be passed to the callback when it is invoked.

flags
One of the following flags.

Flags	Description
WFAppDisplayBorder	Draws a red border around the captured application
WFAppDisplayCursor	Capture the display cursor

Returns

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE**.**

WFReleaseAppDisplay

Stops capturing the contents of an application window, which was previously initiated by calling WFCaptureAppDisplay.

VOID WINAPI **WFReleaseAppDisplay**(

IN HWND hwnd);

hWndThe

window handle of the application for which to stop capturing.

How to detect WFAPI is installed on the system and functioning properly.

Sample

```
#include <tchar.h>
#include <stdio.h>
#include "wfapi.h"

/*Change this to WFAPI for 32 bit*/
#define WFAPILIB _T("WFAPI64.DLL")

/* WFFreeMemory function pointer*/
typedef VOID ( WINAPI *PFnWFFreeMemory) (PVOID);
/* WFSendMessage function pointer*/
typedef BOOL ( WINAPI *PFnWFSendMessage)
(IN  HANDLE hServer,
 IN  DWORD  SessionId,
 IN  LPWSTR pTitle,
 IN  DWORD  TitleLength,
 IN  LPWSTR pMessage,
 IN  DWORD  MessageLength,
 IN  DWORD  Style,
 IN  DWORD  Timeout,
 OUT DWORD* pResponse,
 IN  BOOL   bWait);
/* WFQuerySessionInformation function pointer*/
typedef BOOL ( WINAPI *PFnWFQuerySessionInformation)
                                (IN  HANDLE       hServer,
                                 IN  DWORD        SessionId,
                                 IN  WF_INFO_CLASS WFInfoClass,
                                 OUT LPWSTR*      ppBuffer,
                                 OUT DWORD*       pBytesReturned);
/* WFEnumerateProcesses function pointer*/
typedef BOOL ( WINAPI *PFnWFEnumerateProcesses)
                                (IN  HANDLE           hServer,
                        IN  DWORD           Reserved,
                        IN  DWORD           Version,
                        OUT PWF_PROCESS_INFOW* ppProcessInfo,
                        OUT DWORD*          pCount);
/* WFEnumerateSessions function pointer*/
typedef BOOL ( WINAPI *PFnWFEnumerateSessions)
                                (IN  HANDLE           hServer,
                       IN  DWORD           Reserved,
                       IN  DWORD           Version,
                       OUT PWF_SESSION_INFOW* ppSessionInfo,
                       OUT DWORD*          pCount);

/*UNICODE version of method call : For ANSI replace W with A*/
PFnWFSendMessage pFnWFSendMessage = NULL;
```

```
char aWFSendMessage[] = "WFSendMessageW";


PFnWFEnumerateSessions pFnWFEnumerateSessions = NULL;
char aWFEnumerateSessions[] = "WFEnumerateSessionsW";

PFnWFEnumerateProcesses pFnWFEnumerateProcesses = NULL;
char aWFEnumerateProcesses[] = "WFEnumerateProcessesW";

PFnWFQuerySessionInformation pFnWFQuerySessionInformation = NULL;
char aWFQuerySessionInformation[] = "WFQuerySessionInformationW";

PFnWFFreeMemory pFnWFFreeMemory = NULL;
char aWFFreeMemory[] = "WFFreeMemory";

VOID freeWFMemory(void * pVoid)
{
	if (pVoid != NULL)
	{
		pFnWFFreeMemory(pVoid);
		pVoid = NULL;
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	HINSTANCE dll;
	DWORD sessionid = WF_CURRENT_SESSION;
	LPWSTR pztitle  =_T("WFAPI");
	LPWSTR pzmessage  =_T("WFAPI Test message");
	DWORD resp = 0;

	/*Trying to load WFAPI dll , expecting to have the library in system search path as set in environment variable ,you
can also change this to a specific location */
	dll = LoadLibrary(WFAPILIB);
	if (dll == NULL )
	{
		_tprintf(_T("Could not locate WFAPI on the system...\n"));
		return 0;
	}
	else
	{
		_tprintf(_T("Found WFAPI on the machine...\n"));
	}


	pFnWFSendMessage = (PFnWFSendMessage) GetProcAddress(dll, aWFSendMessage);
	pFnWFFreeMemory = (PFnWFFreeMemory) GetProcAddress(dll, aWFFreeMemory);
	pFnWFEnumerateSessions = (PFnWFEnumerateSessions) GetProcAddress(dll, aWFEnumerateSessions);
	pFnWFEnumerateProcesses = (PFnWFEnumerateProcesses) GetProcAddress(dll, aWFEnumerateProcesses);
	pFnWFQuerySessionInformation = (PFnWFQuerySessionInformation) GetProcAddress(dll,
aWFQuerySessionInformation);

	/* Checking for Non NULL function pointer*/
	if((pFnWFSendMessage != NULL) &&
```

```
		(pFnWFFreeMemory != NULL) &&
		(pFnWFEnumerateSessions != NULL) &&
		(pFnWFEnumerateProcesses != NULL) &&
		(pFnWFQuerySessionInformation != NULL))
	{
		/*Send message to current session and current server*/
		if(pFnWFSendMessage(WF_CURRENT_SERVER_HANDLE,
			sessionid,
			pztitle,
			(_tcslen(_T("WFAPI")) + 1) * sizeof(TCHAR),
			pzmessage,
			(_tcslen(_T("WFAPI Test message")) + 1) * sizeof(TCHAR),
			MB_OK,
			0 ,
			&resp,
			FALSE))
		{
			_tprintf(_T("Message could be send using WFAPI API... \n"));
		}

	}
	FreeLibrary(dll);

	return 0;
}
```

Session Monitoring and Control (SMC)

Introduction

This section contains the Application Programming Interface (API) for the Session Monitoring and Control. The Session Monitoring and Control APIs allow the user mode applications to obtain information about ICA sessions and/or exercise control over bandwidth used by ICA sessions.

The unit of measurement for all bandwidth readings is bits per sec.

Nomenclature

The following data types are assumed to be defined in the following sections:

WCHAR, a 16 bit UNICODE character

UINT32, a 32 bit unsigned integer

UINT64, a 64 bit unsigned integer

Server Functions

WFSmcGetVersionSupported

```
DWORD WINAPI WFSmcGetVersionSupported (
        IN      HANDLE          hServer
        OUT     UINT32*         pVersionSupported,
        OUT     UINT32*         pFeaturesSupported
        );
```

Description

Returns the SMC API version number and the features supported on the current server. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
pVersionSupported	Address of a UINT32 which receives the API version number. The SMC API will be backward compatible with version numbers less than this value. The lowest version number is 1
pFeaturesSupported	Address of a UINT32 which receives a mask containing flags indicating which features of the API are supported. The features supported are dependent upon licensing conditions. The availability of each of the API functions in the SMC API set depends on the values returned through this parameter. The Required Compatibility list presented with each of the APIs indicates the flags required for that API to be exposed. These can be a combination of the following values SMC_API_COMPATIBILITY_XP SMC_API_COMPATIBILITY_XPE_FR1

Return *Values*

ERROR_SUCCESS	The function completed successfully. pVersionSupported points to a UINT32 containing the version number of the SMC API. pFeaturesSupported points to a UINT32 mask defining the feature level.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_PARAMETER	Either of the parameters was NULL.

WFSmcGetServerData

```
DWORD WINAPI WFSmcGetServerData (
        IN      HANDLE                  hServer,
        IN      INT32                   cbData,
        OUT     PWFSMC_SERVER_DATA      pData,
        IN      UINT32                  cbServerVCData,
        OUT     PWFSMC_SERVER_VC_DATA   pServerVCData,
        IN OUT  UINT32*                 pcVCs
        );
```

Description

Returns cumulative totals for the server. See the WFSMC_SERVER_DATA definition for details. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
cbData	The size in bytes of the following structure used for server data.
pData	Address of a PWFSMC_SERVER_DATA structure to be filled in by the call.
cbServerVCData	The size, in bytes, of the following buffer used for an array of structures for the VC specific data.
pServerVCData	Address of an array of WFSMC_SERVER_VC_DATA structs to be filled in by the call.
pcVCs	[IN] Count of entries in the preceding array. [OUT] After the function has returned, this parameter contains the count of entries updated in the array pointed to by pServerVCData. If pServerVCData was NULL or the array was not large enough, the function returns ERROR_MORE_DATA and this parameter contains the required number of entries.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_NOT_ENOUGH_MEMORY	Not enough memory was available to process this request
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_MORE_DATA	There was insufficient space to return all the Virtual Channel data, pcVCs now contains a count of how many there are.
ERROR_INVALID_PARAMETER	Either cbData was less than the size of a Ctx_Smc_Server_Data structure, pData or pcVCs was NULL.
ERROR_NOT_SUPPORTED	The version number passed in pData->VersionNumber is not supported.
OTHER	Some other error occurred internally when trying to access the data.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Session Functions

WFSmcOpenSessionHandle

```
INT WINAPI WFSmcOpenSessionHandle(
        IN      HANDLE          hServer,
        IN      UINT32          SessionId,
        OUT     PHANDLE         phSession
        );
```

Description

Returns a handle to the caller which uniquely identifies a session in the context of this API set.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
SessionId	Session Id of the session for which the handle is required.
phSession	Address of a HANDLE, which receives a handle to the session. This handle is valid in the context of this API set. When the caller has finished with this handle, it should be closed by a call to the WFSmcCloseSessionHandle function.

Returns

ERROR_SUCCESS	The function completed successfully. The memory pointed at by the phSession parameter contains a valid handle.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_ACCESS	The specified connection was inaccessible
ERROR_INVALID_PARAMETER	A NULL value was passed as the phSession parameter
ERROR_NOT_ENOUGH_MEMORY	Not enough memory was available to process this request

WFSmcCloseSessionHandle

```
INT WINAPI WFSmcCloseSessionHandle(
    IN HANDLE hServer,
    IN HANDLE hSession
);
```

Description

Invalidates a session handle, which was obtained through a previous call to the WFSmcOpenSessionHandle function. After a handle is invalidated, calls to the other functions in the API set will return ERROR_INVALID_HANDLE. This function should be called when a handle is no longer required since it allows the DLL to internally release resources used by the handle.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to invalidate. This handle must have been created by a call to the WFSmcOpenSessionHandle function.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter was invalid.

WFSmcGetSessionVirtualChannelNames

```
INT
WINAPI
WFSmcGetSessionVirtualChannelNames (
        IN      HANDLE                          hServer,
        IN      HANDLE                          hSession,
        IN      UINT32                          cbSessionVCNames,
        OUT     PWF_SMC_SESSION_VC_NAME         pSessionVCNames,
        IN OUT UINT32*                          pcSessionVCNames
 );
```

Description

Returns the Virtual Channel names in use for a session. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function.
cbSessionVCNames	The size in bytes of the buffer pointed at by pSessionVCNames used for an array of structures of VC specific data.
pSessionVCNames	Address of an array of WF_SMC_SESSION_VC_NAME structs to be filled in by the call.
pcSessionVCNames	[IN] Count of entries in the preceding array. [OUT] After the function has returned, this parameter contains the count of entries updated in the array pointed to by pSessionVCNames. If pSessionVCNames was NULL or the array was not large enough, the function returns ERROR_MORE_DATA and this parameter contains the required number of entries.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_DATA	The session is valid, but the virtual channel names for the session are not available.
ERROR_MORE_DATA	The buffer pointed at by the pSessionVCNames was of insufficient size to store all the Virtual Channel names in use.
ERROR_NOT_READY	The virtual channels for this session have not completed initialization yet.
ERROR_INVALID_PARAMETER	An invalid parameter was passed. NULL is an invalid value for the pcSessionVCNames parameter.
ERROR_INVALID_HANDLE	The session handle specified by the hSession parameter is invalid.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

Set the VersionNumber data member of structure WF_SMC_SESSION_VC_NAME = SMC_STRUCTURE_VERSION_V1 before calling the API

WFSmcGetSessionData

```
INT WINAPI WFSmcGetSessionData (
        IN      HANDLE                  hServer,
        IN      HANDLE                  hSession,
        IN      UINT32                  cbData,
        OUT     PWF_SMC_SESSION_DATA    pData,
        IN      UINT32                  cbVCData,
        OUT     PWF_SMC_SESSION_VC_DATA pVCData,
        IN OUT  UINT32*                  pcVC
        );
```

Description

Returns the bandwidth usage, compression ratios and latency values for the session, as well as the bandwidth used by the virtual channels in the session if required. The bandwidth usage for the virtual channels is returned as an array of WFSmc_Session_VC_Data structs. Each entry in the array corresponds to the virtual channel returned by a call to the WFSmcGetSessionVirtualChannelNames function. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function
cbData	The size in bytes of the following structure
pData	Address of a WF_SMC_SESSION_DATA structure to return session counters in. Currently, the VersionNumber member of this structure must contain 1.
cbVCData	The size in bytes of the buffer containing the array of WF_SMC_SESSION_VC_DATA structures.
pVCData	Address of an array of WF_SMC_SESSION_VC_DATA structs to be filled in by the call.
pcVC	[IN] Count of entries in the preceding array. [OUT] After the function has returned, this parameter contains the count of entries updated in the array pointed to by pVCData. If pVCData was NULL, or the array was not large enough, the function returns ERROR_MORE_DATA, which contains the required

	number of entries.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_MORE_DATA	There was insufficient space to return all the Virtual Channel data, pcVC now contains a count of how many there are.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid.
ERROR_INVALID_PARAMETER	Either cbData was less than the size of a WFSMC_SESSION_DATA structure, pData or pcVC was NULL.
ERROR_NOT_SUPPORTED	The version number passed in pData->VersionNumber is not supported.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

Set the VersionNumber data member of structures WF_SMC_SESSION_VC_NAME & WF_SMC_SESSION_DATA = SMC_STRUCTURE_VERSION_V1 before calling the API

WFSmcGetSessionDataV2

```
INT WINAPI WFSmcGetSessionDataV2 (
    IN      HANDLE                      hServer,
    IN      HANDLE                      hSession,
    IN      UINT32                      cbData,
    OUT     pCtx_Smc_Session_Data_V2    pData,
    IN      UINT32                      cbVCData,
    OUT     pCtx_Smc_Session_VC_Data_V2 pVCData,
    IN OUT  UINT32*                     pcVC
    );
```

Description

Returns the bandwidth usage, compression ratios and latency values for the session, as well as the bandwidth used by the virtual channels in the session if required. The bandwidth usage for the virtual channels is returned as an array of Ctx_Smc_Session_VC_Data_V2 structures. Each entry in the array corresponds to the virtual channel returned by a call to the WFSmcGetSessionVirtualChannelNames function. This API is supported only on the current server.

This api returns Ctx_Smc_Session_Data_V2 per connection. As per multistream ICA we can have 4 connections per session.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function.
cbData	The size in bytes of the following structure.
pData	Address of a pCtx_Smc_Session_Data_V2 array of size 4 to return session counters in. Currently, the VersionNumber member of this structure must contain 2.
cbVCData	The size in bytes of the buffer containing the array of WF_SMC_SESSION_VC_DATA structures.
pVCData	Address of an array of Ctx_Smc_Session_VC_Data_V2 structs to be filled in by the call.

pcVC	[IN] Count of entries in the preceding array. [OUT] After the function has returned, this parameter contains the count of entries updated in the array pointed to by pVCData. If pVCData was NULL, or the array was not large enough, the function returns ERROR_MORE_DATA, which contains the required number of entries.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_MORE_DATA	There was insufficient space to return all the Virtual Channel data, pcVC now contains a count of how many there are.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid.
ERROR_INVALID_PARAMETER	Either cbData was less than the size of a WFSMC_SESSION_DATA structure, pData or pcVC was NULL.
ERROR_NOT_SUPPORTED	The version number passed in pData->VersionNumber is not supported.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

**Set the VersionNumber data member of structures CTX_SMC_SESSION_DATA_V2 & CTX_SMC_SESSION_VC_DATA_V2 = SMC_STRUCTURE_VERSION_V2 before calling the API

WFSmcGetSessionLatencyData

```
INT WINAPI WFSmcGetSessionLatencyData {
        IN      HANDLE      hServer,
        IN      HANDLE      hSession,
        OUT     UINT32*     pLastLatency,
        OUT     UINT32*     pAverageLatency,
        OUT     UINT32*     pRoundTripDeviation,
        OUT     UINT32*     pTickCountLastUpdateTime
        );
```

Description

Returns the latency information for a session, a NULL pointer for any of the UINT32 pointers is ignored. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function
pLastLatency	Address of a UINT32 which receives the value of the last recorded latency reading (ms)
pAverageLatency	Address of a UINT32 which receives the value of the average latency over the life of the session (ms)
pRoundTripDeviation	Address of a UINT32 which receives the round trip deviation for this session
pTickCountLastUpdateTime	Address of a UINT32 which receives the tick count of the last update time for the latency data.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid.
ERROR_NOT_READY	Latency information is not available for this session. Try again later.

Required Compatibility

SMC_API_COMPATIBILITY_XP

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

Not all ICA sessions support gathering of latency information. Only those clients with a version number of 6.0.910 or later, support gathering of latency information. In addition, depending on the setting used on the client for mouse feedback, the latency information for some clients will take a longer time to become available than for other clients.

WFSmcGetSessionBandwidthLimits

```
INT WINAPI WFSmcGetSessionBandwidthLimits (
        IN      HANDLE          hServer,
        IN      HANDLE          hSession,
        IN      UINT32          cEntries,
        IN      UINT32*         pVCIDs,
        OUT     UINT32*         pVCLimits
        );
```

Description

Returns the bandwidth limit settings for a session. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function
cEntries	Count of entries in the array pointed at by pVCIDs
pVCIDs	Pointer to an array of ids of size cEntries, containing the ids of the VCs for which bandwidth limits are required.
pVCLimits	Pointer to an array of UINT32 in which to return the bandwidth limit values that match the Ids passed in pVCIDs. Bandwidth limits are expressed in kilobits per second.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_PARAMETER	Either one or both of the pointers pVCIDs or pVCLimits is NULL.
ERROR_INVALID_DATA	One or more of the ids in the array pointed to by

	pVCIDs does not match a known Id.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

The bandwidth limits for one or more VCs may be requested by the call. The cEntries specifies how many. The Ids must match the values obtained by a call to WFSmcGetSessionVirtualChannelNames() with the exception that an id value of 0xFFFFFFFF specifies the bandwidth limit of the whole session.

WFSmcSetSessionBandwidthLimits

```
INT WINAPI WFSmcSetSessionBandwidthLimits (

        IN HANDLE   hServer,
        IN HANDLE   hSession,
        IN UINT32   cEntries,
        IN UINT32*  pVCIDs,
        IN UINT32*  pVCLimits
        );
```

Description

Sets the bandwidth limits for a session. This API is supported only on the current server.

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required to set. This handle must have been obtained using a call to the WFSmcOpenSessionHandle function
CEntries	Count of entries in the array pointed at by pVCIDs
pVCIDs	Pointer to an array of ids of size cEntries, containing the ids of the VCs for which bandwidth limits are required.
pVCLimits	Pointer to an array of UINT32 values which is used to set the bandwidth limit values that match the Ids passed in pVCIDs. Bandwidth limits are expressed in kilobits per second.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_PARAMETER	Either the pointer pVCIDs or pVCLimits is NULL.
ERROR_INVALID_DATA	One or more of the ids in the array pointed to by pVCIDs does not match the known
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid
OTHER	There was some other error internally when trying to set the limits.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

The bandwidth limits for one or more VCs may be set by the call. The cEntries specifies how many. The Ids must match the values obtained by a call to WFSmcGetSessionVirtualChannelNames() with the exception that an id value of 0xFFFFFFFF specifies the bandwidth limit of the whole session.

WFSmcGetSessionPriorities

```
INT WINAPI WFSmcGetSessionPriorities(
        IN  HANDLE      hServer,
        IN  HANDLE      hSession,
        IN  UINT32      cEntries,
        IN  UINT32*     pVCIDs,
        OUT UINT32*     pVCPriorities
        );
```

Description

Returns the VC priority settings for this session. This API is supported only on the current server.

Where,

Priorities are from the set

#define CTX_SESSION_VC_PRIORITY_HIGHEST 0

#define CTX_SESSION_VC_PRIORITY_HIGH 1

#define CTX_SESSION_VC_PRIORITY_NORMAL 2

#define CTX_SESSION_VC_PRIORITY_LOW 3

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle should have been obtained using a call to the WFSmcOpenSessionHandle function
cEntries	Count of entries in the array pointed at by pVCIDs
pVCIDs	Pointer to an array of ids of size cEntries, containing the ids of the VCs for which priority settings are required.
pVCPriorities	Pointer to an array of UINT32 in which to return the VC priority values that match the Ids passed in pVCIDs.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_PARAMETER	Either the pointer pVCIDs or pVCPriorities is NULL.
ERROR_INVALID_DATA	One or more of the ids in the array pointed to by pVCIDs does not match the known VCs.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

The priorities for one or more VCs may be requested by the call. The cEntries specifies how many. The Ids must match the values obtained by a call to WFSmcGetSessionVirtualChannelNames().

WFSmcSetSessionPriorities

```
INT
WINAPI
WFSmcSetSessionPriorities(
IN HANDLE  hServer,
        IN HANDLE       hSession,
        IN UINT32       cEntries,
        IN UINT32*      pVCIDs,
        IN UINT32*      pVCPriorities
        );
```

Description

Sets the VC priority settings for this session. This API is supported only on the current server.

Where,

Priorities are from the set

```
#define CTX_SESSION_VC_PRIORITY_HIGHEST     0
#define CTX_SESSION_VC_PRIORITY_HIGH        1
#define CTX_SESSION_VC_PRIORITY_NORMAL      2
#define CTX_SESSION_VC_PRIORITY_LOW         3
```

Parameters

hServer	A handle to a server that is running XenApp Server. Specify WF_CURRENT_SERVER_HANDLE to use the current server.
hSession	Handle to the session for which the data is required. This handle should have been obtained using a call to the WFSmcOpenSessionHandle function
cEntries	Count of entries in the array pointed at by pVCIDs
pVCIDs	Pointer to an array of ids of size cEntries, containing the ids of the VCs for which priorities are to be set.
pVCPriorities	Pointer to an array of UINT32 containing the priority values to be set for the Ids passed in pVCIDs.

Returns

ERROR_SUCCESS	The function completed successfully.
ERROR_CALL_NOT_IMPLEMENTED	A required entry in the Required Compatibility list wasn't met, and the call has been disabled.
ERROR_SERVICE_NOT_ACTIVE	A service that the dll depends on was not started, or is unavailable.
ERROR_INVALID_PARAMETER	Either the pointer pVCIDs or pVCPriorities is NULL.
ERROR_INVALID_DATA	One or more of the ids in the array pointed to by pVCIDs does not match the known VCs or a priority in pVCPriorities is out of range.
ERROR_INVALID_HANDLE	The handle specified by the hSession parameter is invalid
Other	There was some other error internally when trying to set the priorities.

Required Compatibility

SMC_API_COMPATIBILITY_XPE_FR1

Remarks

The priorities for one or more VCs may be set by the call. The cEntries specifies how many. The Ids must match the values obtained by a call to WFSmcGetSessionVirtualChannelNames().

Appendix A: Structures

WF_SMC_SERVER_DATA

typedef struct

```
{
UINT32          VersionNumber;

UINT64          TotalBytesSentPreCompression;
UINT64          TotalBytesSentPostCompression;

UINT64          TotalBytesReceivedPreExpansion;
UINT64          TotalBytesReceivedPostExpansion;

UINT32          BandwidthSentBitsPerSec;
UINT32          BandwidthReceivedBitsPerSec;

UINT32          CompressionNumeratorSent;
UINT32          CompressionDenominatorSent;

UINT32          CompressionNumeratorReceived;
UINT32          CompressionDenominatorReceived;

} WF_SMC_SERVER_DATA, * PWF_SMC_SERVER_DATA;
```

Members

VersionNumber	Version Number of this structure, value of 1
TotalBytesSentPreCompression	Cumulative total bytes sent from all Virtual Channels before compression since the server was last rebooted
TotalBytesSentPostCompression	Cumulative total of bytes sent to the wire since the server was last rebooted. This does not include TCP/IP or wire packet overhead.
TotalBytesReceivedPreExpansion	Cumulative total of bytes received from the wire since the server was last rebooted. This does not include TCP/IP or wire packet overhead.
TotalBytesReceivedPostExpansion	Cumulative total of bytes passed to Virtual Channels after expansion since the server was last rebooted.
BandwidthSentBitsPerSec	The ICA bandwidth sent from the server to all clients.
BandwidthReceivedBitsPerSec	The ICA bandwidth received at the server from all

	clients.
CompressionNumeratorSent	The numerator for the compression ratio for server to client traffic
CompressionDenominatorSent	The denominator for the compression ratio for server to client traffic
CompressionNumeratorReceived	The numerator for the compression ratio for client to server traffic
CompressionDenominatorReceived	The denominator for the compression ratio for client to server traffic

WF_SMC_SERVER_VC_DATA

```
typedef struct
{
  UINT32       VersionNumber;

  UINT32       InUseCount;

  UINT32       BandwidthSentBitsPerSecPostCompressionEstimate;
  UINT64       BytesSent;

  UINT32       BandwidthReceivedBitsPerSecPreExpansionEstimate;
  UINT64       BytesReceived;

  WCHAR        rgwchName[8];

} WF_SMC_SERVER_VC_DATA, * PWF_SMC_SERVER_VC_DATA;
```

Members

VersionNumber	Version Number of this structure, value of 1.
InUseCount	Count of current sessions sending or receiving data through this Virtual Channel.
BandwidthSentBitsPerSecPostCompressionEstimate	The estimated bandwidth on this virtual channel from the server to all clients.
BytesSent	Cumulative total of bytes sent through this Virtual Channel.
BandwidthReceivedBitsPerSecPreExpansionEstimate	The estimated bandwidth on this virtual channel from all clients to the server.
BytesReceived	Cumulative total of bytes received through this Virtual Channel.
rgwchName	The internal name of this Virtual Channel, which is less than or equal to eight characters including the terminating NULL.

WF_SMC_SESSION_DATA

```
typedef struct
{
   UINT32     VersionNumber;

   UINT32     BandwidthSentBitsPerSecond;
   UINT32     BytesSentPreCompression;
   UINT32     BytesSentPostCompression;

   UINT32     BandwidthReceivedBitsPerSecond;
   UINT32     BytesReceivedPreExpansion;
   UINT32     BytesReceivedPostExpansion;

   UINT32     CompressionNumeratorSent;
   UINT32     CompressionDenominatorSent;
   UINT32     CompressionNumeratorReceived;
   UINT32     CompressionDenominatorReceived;

   UINT32     LastClientLatency;
   UINT32     AverageClientLatency;
   UINT32     RoundTripDeviation;
   UINT32     LastUpdateTime;

   UINT32     SessionBandwidthLimitKBitsPerSecond;

   UINT32     OutputSpeedInBitsPerSecond;
   UINT32     InputSpeedInBitsPerSecond;

} WF_SMC_SESSION_DATA, * PWF_SMC_SESSION_DATA;
```

Members

VersionNumber	Version Number of this structure, value of 1
BandwidthSentBitsPerSecond	The current bandwidth of this session for sent data.
BytesSentPreCompression	The total bytes sent by this session before compression has occurred
BytesSentPostCompression	The total bytes sent by this session after compression has occurred.
BandwidthReceivedBitsPerSecond	The current bandwidth of this session for received data.
BytesReceivedPreExpansion	The total bytes received by this session

	before expansion of the data has occurred.
BytesReceivedPostExpansion	The total number of bytes received by this session after expansion of the data has occurred.
CompressionNumeratorSent	The current numerator of compression for bytes sent.
CompressionDenominatorSent	The current denominator of compression for bytes sent.
CompressionNumeratorReceived	The current numerator of expansion of bytes received.
CompressionDenominatorReceived	The current denominator of expansion of bytes received.
LastClientLatency	Last recorded latency reading for this session.
AverageClientLatency	Average calculated latency for this session.
RoundTripDeviation	Round trip deviation for this session.
LastUpdateTime	The tick count at the last update of latency values.
SessionBandwidthLimitKBitsPerSecond	The bandwidth cap on this session in K Bits / Second
OutputSpeedInBitsPerSecond	The output line speed from server to client for a session in bits per second
InputSpeedInBitsPerSecond	The input line speed from client to sever for a session in bits per second

Remarks

For all latency counters, a value of 0xFFFFFFFF means it is invalid.

WF_SMC_SESSION_VC_DATA

```
typedef struct
{
UINT32        VersionNumber;
UINT32        idVC;
UINT32        PreReducerBitsPerSecond;
UINT32        BandwidthSentPostCompressionEstimate;
UINT32        BytesSentTotal;
UINT32        PostExpanderBitsPerSecond;
UINT32        BandwidthReceivedPreExpansionEstimate;
UINT32        BytesReceivedTotal;
} WF_SMC_SESSION_VC_DATA, * PWF_SMC_SESSION_VC_DATA;
```

Members

VersionNumber	Version number of this structure, value of 1.
idVC	The id number of this Virtual Channel for which data is to be retrieved
PreReducerBitsPerSecond	The bit rate passing into the reducer before compression has occurred.
BandwidthSentPostCompressionEstimate	The estimated sent bandwidth on the wire for this VC. This value takes into account the output session compression ratio.
PostExpanderBitsPerSecond	The bit rate of the channel after expansion of the data stream from the client has taken place.
BandwidthReceivedPreExpansionEstimate	The estimated received bandwidth on the wire for this VC. This is calculated using the session compression ratio.
BytesReceivedTotal	Total bytes received on the virtual channel

CTX_SMC_SESSION_DATA_V2

```
typedef struct
{
   UINT32     VersionNumber;
   UINT32     QosLevel;
   UINT32     BandwidthSentBitsPerSecond;
   UINT32     BytesSentPreCompression;
   UINT32     BytesSentPostCompression;
   UINT32     BandwidthReceivedBitsPerSecond;
   UINT32     BytesReceivedPreExpansion;
   UINT32     BytesReceivedPostExpansion;
   UINT32     CompressionNumeratorSent;
   UINT32     CompressionDenominatorSent;
   UINT32     CompressionNumeratorReceived;
   UINT32     CompressionDenominatorReceived;
   UINT32     LastClientLatency;
   UINT32     AverageClientLatency;
   UINT32     RoundTripDeviation;
   UINT32     LastUpdateTime;
   UINT32     SessionBandwidthLimitKBitsPerSecond;
   UINT32     OutputSpeedInBitsPerSecond;
   UINT32     InputSpeedInBitsPerSecond;
   UINT32     IsConnectionActive;
   UINT32          IsPrimaryStream;

} Ctx_Smc_Session_Data_V2, *pCtx_Smc_Session_Data_V2;
```

Members

VersionNumber	Version Number of this structure, value of 2
QosLevel	This is connection-within session and corresponds to QoSLevel that can be between 0 – 3.
BandwidthSentBitsPerSecond	The current bandwidth of this session-connection for sent data.
BytesSentPreCompression	The total bytes sent by this session-connection before compression has occurred
BytesSentPostCompression	The total bytes sent by this session-connection after compression has occurred.
BandwidthReceivedBitsPerSecond	The current bandwidth of this session-connection for received data.

BytesReceivedPreExpansion	The total bytes received by this session-connection before expansion of the data has occurred.
BytesReceivedPostExpansion	The total number of bytes received by this session-connection after expansion of the data has occurred.
CompressionNumeratorSent	The current numerator of compression or bytes sent for this session-connection.
CompressionDenominatorSent	The current denominator of compression for bytes sent for this session-connection.
CompressionNumeratorReceived	The current numerator of expansion of bytes received for this session connection.
CompressionDenominatorReceived	The current denominator of expansion of bytes received for this session-connection.
LastClientLatency	Last recorded latency reading for this session-connection
AverageClientLatency	Average calculated latency for this session-connection.
RoundTripDeviation	Round trip deviation for this session-connection.
LastUpdateTime	The tick count at the last update of latency values for this session-connection.
SessionBandwidthLimitKBitsPerSecond	The bandwidth cap on this session-connection in K Bits / Second
OutputSpeedInBitsPerSecond	The output line speed from server to client for a session-connection in bits per second
InputSpeedInBitsPerSecond	The input line speed from client to sever for a session-connection in bits per second
IsConnectionActive	Non-zero if this connection in the session is active otherwise it is 0.
IsPrimaryStream	Non-zero if this connection is the primary

	connection within the session otherwise it is 0.

CTX_SMC_SESSION_VC_DATA_V2

```
typedef struct
{
   UINT32     VersionNumber;
   UINT32     idVC;
   WCHAR    rgwchName[8];
   UINT32     QosLevel;
   UINT32     PreReducerBitsPerSecond;
   UINT32     BandwidthSentPostCompressionEstimate;
   UINT32     BytesSentTotal;
   UINT32     PostExpanderBitsPerSecond;
   UINT32     BandwidthReceivedPreExpansionEstimate;
   UINT32     BytesReceivedTotal;
   UINT32     BandwidthLimitKBitsPerSecond;
} Ctx_Smc_Session_VC_Data_V2, *pCtx_Smc_Session_VC_Data_V2;
```

Members

VersionNumber	Version number of this structure, value of 2.
idVC	The id number of this Virtual Channel for which data is to be retrieved
rgwchName	Name of the virtual channel
QosLevel	QosLevel or connection-within the session on which the virtual channel is running.
PreReducerBitsPerSecond	The bit rate passing into the reducer before compression has occurred.
BandwidthSentPostCompressionEstimate	The estimated sent bandwidth on the wire for this VC. This value takes into account the output session compression ratio.
PostExpanderBitsPerSecond	The bit rate of the channel after expansion of the data stream from the client has taken place.
BandwidthReceivedPreExpansionEstimate	The estimated received bandwidth on the wire for this VC. This is calculated using the session compression ratio.
BytesReceivedTotal	Total bytes received on the virtual channel

WF_SMC_SESSION_VC_NAME

```
typedef struct
{
    UINT32          VersionNumber,
    UINT32          idVC,
    WCHAR           rgwchName[8]
} WF_SMC_SESSION_VC_NAME, * PWF_SMC_SESSION_VC_NAME;
```

Members

VersionNumber	Version number of this structure, value of 1.
idVC	The id number of this VC to be used in other calls for this session.
rgwchName	The null terminated WCHAR name of the Virtual Channel.

Windows Monitoring API

Introduction

This section contains the host side of the Windows Monitoring API. This API allows the creation of solutions that synchronize the visual aspects of an application that runs on a server with corresponding visual elements that are running on the Citrix Plug-in. The host component of this API allows users to subscribe to tracking of specific windows. When a window's regions change or the window is deleted, the component delivers a message to the user's predefined mailslot. The following API works on win2k8r2 and win 7 when using legacy graphics. It is not supported on the new graphics system.

Key points

- This API provides efficient tracking of windows on the server
- Provides methods for synchronizing with the client desktop display through the Virtual Channel SDK.
- Provides an improved visual experience and support to third-party applications for better ICA integration.

Using the API

Getting Started

Headers: wfapi.h

Libraries: wdica.sys, vdtw30.dll, and wfapi.dll

Architecture

The API provides two distinct components with their own architectures: host-side and client-side. The host component is a part of the WinFrame API, and provides updates on tracked windows. You can then communicate this data to Citrix Plug-in so as to synchronize window positions. The client-side component in the Virtual Channel SDK then allows third parties to synchronize with the ICA window. It provides them with information about the ICA window's dimensions and handle, as well as whether it is panning or scaling. As a whole, the API allows

third-party applications to better integrate with ICA and provide a better visual experience.

The host side consists of two distinct parts split between user and kernel mode. Users subscribe to the system and start tracking windows using the user mode API available in wfapi.dll. Chosen windows are then tracked at the kernel mode level. This provides faster updates and access to more information about the windows (that is, the visible regions of the entire window, not just its client area.) These updates are then sent to subscribers' assigned mailslots.

Sample

In addition to the following sample, there is another example found in the TESTALL example program included with this API

This sample depicts a simple sequence of events that demonstrates the use of each of the API functions. A user first subscribes and is given a mailslot handle. Tracking then starts on a window, and one update message is read from the mailslot. After that, tracking stops on the window, and the user unsubscribes from window tracking. Finally, the user closes the mailslot (such as when it is no longer needed).

```
WF_RESPONSE_METHOD     commMethod;
HANDLE             myMailSlot;
UINT               myId;
WF_WINDOW_DATA message;
WINDOWINFO  windowInfo;

/*Fill in communication structure and subscribe*/
commMethod.Type = WFMailslotResponse;
commMethod.Params.MailSlot.WaitTime = MAILSLOT_WAIT_FOREVER;
commMethod.Params.MailSlot.pSecurityAttributes = NULL;
WFSubscribe(&myId, &commMethod);

 /*My id is filled for you, and identifies you when calling any of the other api functions*/
/*A mailslot is created for you, and its handle passed back */
myMailSlot = commMethod.Params.MailSlot.hSlot;

/*Start tracking a window by passing in your id and the window to          track*/
WFStartTrackingWindow(myId, hwnd);

/*Changes along with the initial status are sent to your mailslot*/
GetMailslotInfo( myMailSlot, NULL, &nextSize, &messageCount, NULL);

/*Read the window message from the mailslot*/
ReadFile(myMailSlot, (LPVOID)&message, sizeof(WF_WINDOW_DATA), &numRead, &ov);

/* Stop tracking the window, stop receiving notifications for this              window */
 WFStopTrackingWindow(myId, hwnd);

 /*Stop all tracking, no notifications of any kind will be sent to your mailslot now*/
WFUnsubscribe(myId);
```

```
/*User is responsible for closing mailslot*/
CloseHandle(myMailSlot);
```

Programming guide

The API as a whole provides better window control for applications that coordinate windows between the host and client desktop. In general, the host uses the WinFrame API component of the API to track windows of interest. The host listens on an assigned mailslot for tracking updates about its windows. These updates are then communicated to the Citrix Plug-in, where they are used to properly position corresponding windows. The Citrix Plug-in uses the client-side portion of the APIs in the Virtual Channel SDK to synchronize its windows with the ICA window. The Citrix Plug-in can be notified when the ICA window changes, and thus make any necessary changes to other third-party applications.

Programming Reference

Functions

Note: all these functions return TRUE upon success. If the function returns FALSE, use the GetLastError function to find out what went wrong.

WFSubscribe

```
BOOL WINAPI WFSubscribe(OUT UINT* pUserId, IN OUT PWF_RESPONSE_METHOD pCommMethod);
```

WFSubscribe is called initially by a user to set up a method of delivery for the window tracking updates.

pUserId

Upon returning, the user receives a unique ID number. Any future calls require the use of this ID number to identify the user.

pCommMethod

This structure describes the communication method the user wants to use to receive tracking updates. Currently, only mailslots are supported. The user supplies the WaitTime and pSecurityAttributes parameters that the SDK uses to create a new mailslot. Upon returning, the structure is populated with a handle to the mailslot hSlot.

WFUnsubscribe

```
BOOL WINAPI WFUnsubscribe(IN UINT UserId);
```

WFUnsubscribe is called when the user no longer wants to make use of the tracking system. Tracking stops for any previously tracked windows by this user and the user's ID becomes invalid. The user is responsible for closing the mailslot.

UserId

Unique user ID previously returned by WFSubscribe.

WFStartTrackingWindow

```
BOOL WINAPI WFStartTrackingWindow(IN UINT Id, IN HWND hWnd);
```

Used to start tracking a window after a user subscribes.

Id
: The user's unique ID number.

hWnd
: Handle of the window to start tracking.

WFStopTrackingWindow

```
BOOL WINAPI WFStopTrackingWindow(UINT Id, HWND hWnd);
```

WFStopTrackingWindow is used to stop tracking a window.

Id
: The user's unique ID number.

hWnd
: Handle of the window to stop tracking.

Structures

WF_MAILSLOT_INFO

Used when subscribing to set the parameters of the mailslot to be created. The user is responsible for setting WaitTime and pSecurityAttributes. Upon successful return, hSlot is populated with the newly created mailslot's handle. *Note: The user is responsible for closing this handle when the mailslot is no longer needed.*

```
typedef struct _WF_MAILSLOT_INFO
{
    HANDLE              hSlot;
    DWORD               WaitTime;
    LPSECURITY_ATTRIBUTES   pSecurityAttributes;
} WF_MAILSLOT_INFO;
```

WF_RESPONSE_METHOD

Used when subscribing to window tracking. Describes how the user wants to be notified of window changes. Currently, only mailslots are supported.

```
typedef struct _WF_RESPONSE_METHOD
{
    WF_RESPONSE_METHOD_TYPE Type;
    union PARAMS
    {
        WF_MAILSLOT_INFO MailSlot;
    } Params;
} WF_RESPONSE_METHOD, * PWF_RESPONSE_METHOD;
```

WF_WINDOW_DATA

Users receive window notifications initially when tracking starts and then when a window's region changes or a window is destroyed. These messages consist of WF_WINDOW_DATA structures. Each message contains at most 17 clipping rectangles for the window (one in EnumRects.arcl and 16 in RclMoreClip. In the case where there are more than 17 clipping rectangles, the remaining clipping rectangles are sent in subsequent messages. If bMore is TRUE, then the user should keep reading messages to get all the clipping rectangles. When the last chunk of clipping rectangles is sent for this specific message, bMore is FALSE.

```
typedef struct _WF_WINDOW_DATA
{
   WF_WINDOW_MESSAGE_TYPE          MessageType;
   HWND                            hWindow;
   RECTL                           RclClient;
   ULONG                           Height;
   ULONG                           Width;
   BYTE                            iDComplexity;
   RECTL                           RclBounds;
   BOOL                            bMore;
   WF_ENUMRECTS                    EnumRects;
   RECTL                           RclMoreClip[WF_NUM_REGION_RECTS];
} WF_WINDOW_DATA, * PWF_WINDOW_DATA;
```

MessageType
: Type of message, such as window update or deletion.

hWindow
: Handle of the window being tracked.

RclClient
: The client area of the window.

Height
: Height of the window.

Width
: Width of the window.

iDComplexity
: Complexity of the window's visible region. See msdn for more information.

RclBounds
: Rectangle that bounds the entire window.

bMore
: Flag indicating whether more region rectangles for this window can be expected in subsequent messages.

EnumRects
: Clipping information. Contains the total number of rectangles in this message and one rectangle.

RclMoreClip
: Up to 16 more clipping rectangles.

WF_ENUMRECTS

Used in the WF_WINDOW_DATA window notifications. Contains the total number of visible region rectangles in this message, as well as one of these rectangles.

```
typedef struct _WF_ENUMRECTS
{
    ULONG  c;
    RECTL  arcl[1];
} WF_ENUMRECTS;
```

Enumerations

WF_RESPONSE_METHOD_TYPE

Type of notification system for receiving window tracking updates, currently only mailslots are supported.

```
typedef enum _WF_RESPONSE_METHOD_TYPE
{
    WFReservedResponse = 0,
    WFMailslotResponse = 1
} WF_RESPONSE_METHOD_TYPE;
```

WF_WINDOW_MESSAGE_TYPE

Type of window notification message, such as for an update or notification of deletion.

```
typedef enum _WF_WINDOW_MESSAGE_TYPE
{
    WFWindowUpdated = 0,
    WFWindowDeleted = 1
} WF_WINDOW_MESSAGE_TYPE;
```

TESTALL

Test results

#	**Test**	**Windows 10**	**Windows 11**	**Windows 2019**
1	Enumerate Sessions	Console	Console	Pass
2	Enumerate Processes	Fail	Fail	Pass
3	WF Version	Pass	Pass	Pass
4	WF Working Directory	Fail	Fail	Pass
5	WF OEM Id	Fail	Fail	No Data
6	WF Session Id	Pass	Pass	Pass
7	WF User Name	Pass	Pass	Pass
8	WF Winstation	Pass	Pass	Pass
9	WF Domain Name	Pass	Pass	Pass
10	WF Connect State	Pass	Pass	Pass
11	WF Client Build	Pass	Pass	Pass
12	WF Client Name	Pass	Pass	Pass
13	WF Client Directory	Pass	Pass	Pass
14	WF Client Product ID	Pass	Pass	Pass
15	WF Client Address	Pass	Pass	Pass
16	WF Client Display	Pass	Pass	Pass
17	WF Client Cache	Pass	Pass	Pass
18	WF Client Drives	Pass	Pass	Pass
19	WF ICA Buffer Length	Pass	Pass	Pass
20	WF Initial Program	Fail	Fail	Pass
21	WF Client Application Session	Session Info	Session Info	Session Info
22	Virtual Channel Test	Pass	Pass	Pass
23	Query User Config - Domain	Fail	Fail	Pass
24	Set User Config – Domain	Fail	Fail	Pass
25	Query User Config - Local	Fail	Fail	Pass
26	Set User Config Local	Fail	Fail	Fail
27	WF Client Info	Pass	Pass	Pass
28	WF Application Info	Pass	Pass	Pass
29	WF User Info	Pass	Pass	Pass
30	WF Session Time	Pass	Pass	Pass
31	WF Enumerate Processes (more info)	Fail	Fail	Pass

32	WF Wait Session Events	Pass	Pass	Pass
33	WF Send Message	Pass	Pass	Pass
34	WF Disconnect	Pass	Pass	Pass
35	WF Disconnectex	Pass	Pass	Pass
36	WF Logoff	Pass	Pass	Pass
37	WF terminate Pro-cess	Pass	Pass	Pass
38	WF Shutdown	Pass	Pass	Pass
39	WF License Model	Pass	Pass	Pass
40	WF GetCtx-SesionKeyList	No list	No List	Pass

CITRIX